¿Qué es el Java String Pool? . Muchas veces cuando trabajamos con Java tenemos que comparar objetos y tipos básicos entre ellos. Si recordamos de otro artículo anterior , para comparar tipos básicos usamos el operador == que es el operador de comparación . Por otro lado si comparamos objetos usamos el método equals(). Hasta aquí todo es perfecto pero existen algunas excepciones que generan dudas. Vamos a hablar un poco de ellas. Para ello partiremos de la clase Persona que tiene una propiedad nombre.

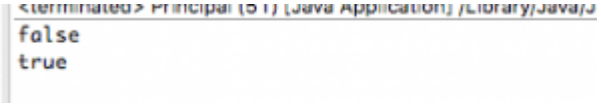

```
package com.arquitecturajava;

public class Persona {

	private String nombre;

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public Persona(String nombre) {
```

```
                super();
                this.nombre = nombre;
        }

        @Override
        public int hashCode() {
                final int prime = 31;
                int result = 1;
                result = prime * result + ((nombre == null) ? 0 :
nombre.hashCode());
                return result;
        }

        @Override
        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (obj == null)
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                Persona other = (Persona) obj;
                if (nombre == null) {
                        if (other.nombre != null)
                                return false;
                } else if (!nombre.equals(other.nombre))
                        return false;
                return true;
        }

}
```

Vamos a comparar dos objetos con el mismo nombre:

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {

		Persona p1= new Persona("pepe");
		Persona p2= new Persona("pepe");

		System.out.println(p1==p2);
		System.out.println(p1.equals(p2));

	}

}
```

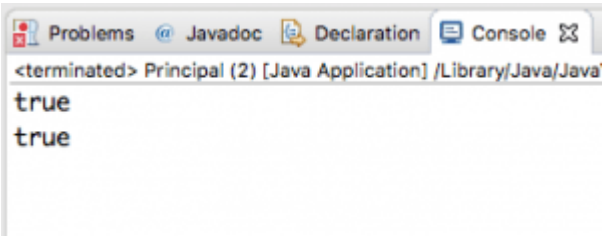
Acabamos de crear dos objetos Persona y compararlos entre ellos .



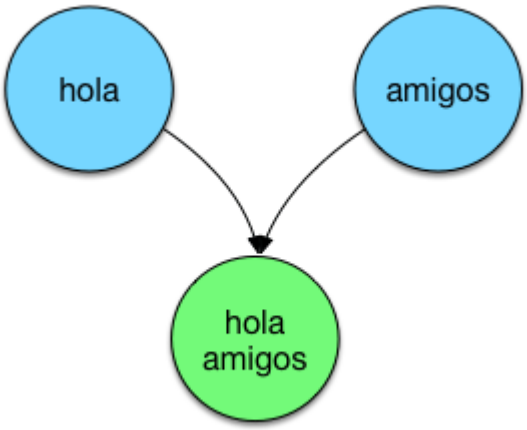
Hemos obtenido el resultado esperado si comparamos con == nos devuelve false son objetos diferentes y si comparamos con equals nos devuelve true ya que el nombre coincide. Vamos a construir el mismo ejemplo pero usando la clase String

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {

		String cadena="hola";
		String cadena2="hola";
		System.out.println(cadena.equals(cadena2));
		System.out.println(cadena==cadena2);

	}

}
```

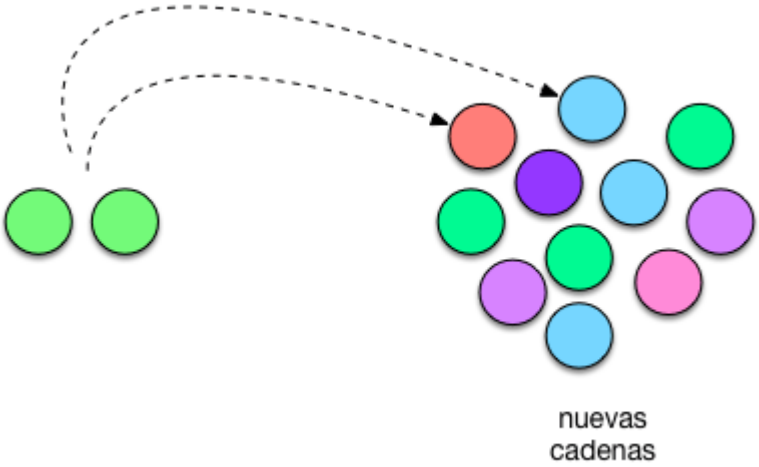
Como podemos ver tenemos dos objetos con la misma cadena de texto . Sin embargo cuando los comparamos con == y equals el resultado no es el que esperábamos los dos devuelven true.



¿Qué es lo que esta pasando realmente? . ¿Hemos entendido bien como funcionan == e equals? . La realidad es que el caso de los Strings es un caso “muy particular”, Recordemos que los Strings son objetos Inmutables , es decir una vez creado un String jamas puede ser modificado. Cada vez que modificamos una cadena la realidad es que se crea una nueva.

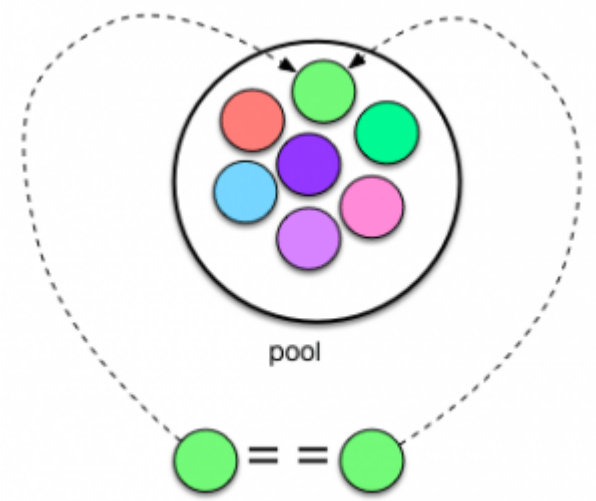


Esto hace que las cadenas sean Threadsafe y puedan ser accedidas por varios hilos de forma simultánea sin generar ningún tipo de problema. Ahora bien todo tiene un precio, si cada vez que modificamos una cadena se crea una nueva, esto implicará que el programa tendrá que almacenar muchas cadenas. La implicación de este enfoque es clara , el consumo de memoria se incrementará ya que hay muchas cadenas iguales.



Java String Pool

Para reducir este problema Java de forma interna implementa el patrón flyweight y genera un pool de Strings que se comparte . De esta forma cada vez que nosotros necesitamos crear una nueva cadena Java revisa si ya existe en el pool, en tal caso nos devuelve una referencia a ella.



De ahí que cuando creamos dos cadenas con el mismo texto el operador == nos devuelva true siempre, ya que apuntan al mismo objeto del pool. Este es el concepto de Java String pool que a veces cuesta tanto entender.

Otros artículos relacionados:

Java Fluid Interface

Java constructores this() vs super()

Java Lambda